Attention: *Upload via Canvas till 23:59 November 29. You should submit the solutions for problem 1 in a Jupyter Notebook file (with presented analytical solutions, code and explanations attached).*

Attention 2: *The next assignment will contain quite a number of theoretical tasks. Solutions will have to be prepared via LaTeX, so no photo or scanned paper-based solutions will be accepted. So, for those of you who did not have an opportunity to work with LaTeX, please install necessary software, or you can use Sharelatex.*

Problem 1

Overall number of points: 11

You have built a robot for navigating in an indoor environment. The environments you consider have flat floors, so that the whole task is always two-dimensional. The robot has a sensing capability, as it can sense a location with respect to special beacons. You have given a robot to a friend, who uses it to map different interiors. To do the mapping, your friend places beacons at the perimeter of the interior, and then moves the robot through some trajectory. Every so often the robot stops, uses its compass to align itself with the direction to the north and then tries to sense the location of the beacons (relative to the robot's current position). Typically, it manages to locate several closest beacons. Your friend has conducted three experiments in three different interiors. You need to recover the location of beacons and the locations of the robot (where the measurements have been taken). From each experiment you get four arrays of measurements: **beacons, robots, x, y** . It is interpreted as follows: while the robot is in the position number **robots[i]**, it senses the beacon number **beacons[i]**, and the beacon is displaced by **x[i]** and **y[i]** from the robot's position. For each robot position, the robot senses several beacons. This data should be sufficient to recover the positions of the robot and of the beacons (up to a global translation, that does not matter for the purposes of your friend).

Hint and instructions: *To download the datasets you can use (with appropriate change of the file name):*

```
import scipy.io as sio

mat_contents = sio.loadmat('task.mat')

beacons = mat_contents['beacons']
robots = mat_contents['robots']
x = mat_contents['x']
y = mat_contents['y']
```

Sub-problem 1

Points: 1

Dataset: task1.mat

Task:

So, firstly, consider the simplest environment (a room with a simple shape), the robot sensor works well (although a certain amount of Gaussian noise is present). The reconstruction can be reduced to solving a simple least-squares problem.

Hint and instructions: use linear least-squares. Avoid using the explicit inversion of matrix. Instead, use **numpy.linalg.lstsq** to solve equations. Note that **numpy.linalg.lstsq** might add a (small) regularization to the system. Try using inversion (**numpy.linalg.inv**) to see how the reconstruction performs without regularization.

Sub-problem 2

Points: 4

Dataset: task2.mat

Task:

This is a more complex environment. On top of that, there are strong electrical currents under the floor, so that the compass on the robot makes significant errors, so that the robot is not perfectly oriented. Thus, in addition to robot positions you would also need to recover robot orientations.

Hint and instructions: you can still try to solve the problem as if the robot is perfectly oriented. You can then use what you obtain as an initialization to the non-linear initialization (angles can be initialized at zero). For the non-linear optimization, you should use non-linear least-squares. You can then either code Levenberg-Marquardt method yourself or figure out how to use **scipy.optimize**. In the latter case, to get the points, your final version must provide the python's routine with a Jacobian computed (analytically) by your code.

Sub-problem 3

Points: 6

Dataset: task3.mat

Task:

This is another complex environment. On top of the complications from the previous experiment, your robot beacon identification system is now working imperfectly, so that every so often (say, in 3% cases), the robot would confuse the number of the beacon that it observes. You have to make your reconstruction algorithm robust to such mistakes.

Hint and instructions: use non-linear optimization **scipy.optimize** in python (you get 2 points for any implementation that produces an accurate reconstruction). You are recommended to supply it with an analytically-computed gradient (+2 points). Use the option **method=BFGS** to make python to use a quasi-Newton BFGS method, play with the number of iterations that are allowed (**options.maxiter**) to see the effect; you can also try to provide the algorithm with the Hessian derived analytically. You get +2 points if your approach devised this way will work faster or comparable with quasi-Newton.